UNIVERSITY of DEBRECEN

Dr. József Sütő
Department of IT Systems and Networks

Embedded Systems Laboratory

Lab 8: OpenCV installation and the Pi camera usage

Project overview

This laboratory presents how to install **OpenCV** with Python 3 on Raspbian Pi 3 B+. OpenCV is an open source machine vision and machine learning library which will be necessary in later laboratories. The main goal of this tutorial is to guide you step-by-step through the installation process. After the camera is available for us, we will read barcodes and QR codes with the aid of OpenCV and the **ZBar library**. We will implement and use a barcode scanner on the Pi. This laboratory content is based on the pyimagesearch website's posts.

The following topics will be covered in this lab:

- Raspberry Pi camera
- OpenCV library
- QR and barcode detection

Technical requirements

The following components are required to complete this lab:

- Raspberry Pi 3 Model B+ (MicroSD card, power supply, keyboard, mouse)
- Pi camera module

OR

- PC
- USB/built-in camera

1. Raspberry Pi camera

- There are two version of the camera module:
  - Standard – take picture in normal light
  - NoIR – camera without infra-red filter to take pictures in the dark
- Connected to the Pi with ribbon cable via the CSI camera port

- The **picamera** package gives us access to the camera through Python
- The Pi Camera module connects directly to the GPU
- h.264 video encoding
- Documentation: https://picamera.readthedocs.io/en/release-1.13/

2. Install requirements on your Pi

The Raspberry Pi requires that, you install a few system packages before you go ahead.

2.1. Update and upgrade all existing packages:

$ sudo apt-get update

$ sudo apt-get upgrade

2.2. Install HDF5 libraries. It is a specific data type which is very popular in machine learning:

$ sudo apt-get install libhdf5-dev libhdf5-serial-dev

$ sudo apt-get install libhdf5-100

2.3. Install Qt for GUI (graphical user interface) support:

$ sudo apt-get install libqtgui4 libqtwebkit4

$ sudo apt-get install libqt4-test python3-pyqt5

2.4. Many operations inside OpenCV (e.g. matrix operations) can be optimized further by installing a few extra dependencies:

$ sudo apt-get install libatlas-base-dev

2.5. We also need to install an image I/O package to load various image file formats from disk:

$ sudo apt-get install libjasper-dev

3. Install "pip" and OpenCV

pip is a package installer for Python. You can use pip to install packages from the Python Package Index or other indexes. Python Package Index (PyPi) is a repository of software for the Python programming language.

3.1. The Python package manager, **"pip"** can be obtained via *wget*:

$ wget https://bootstrap.pypa.io/get-pip.py

$ sudo python3 get-pip.py

3.2. Install OpenCV with pip. The version of OpenCV that will get installed via pip does not include non-free algorithms such as SIFT, SURF, and other patented algorithms. Using pip is great way to install OpenCV if you won't need to run programs containing non-free algorithms. If non-free algorithms are necessary, you'll need to complete the full OpenCV.

$ sudo pip3 install opencv-contrib-python==4.1.0.25

3.3. Test that every software is working properly. Open a Python shell (Python3) in the Terminal and issue the following commands:

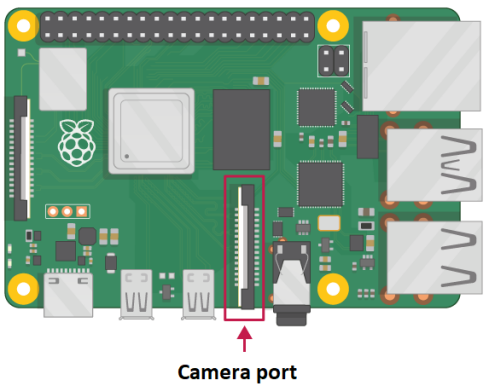
>>> import cv2

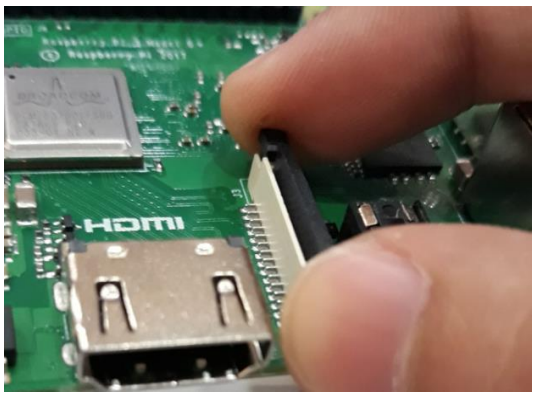
>>> cv2.__vesion__

4. Attach the camera to your Pi (probably your camera is already connected!)

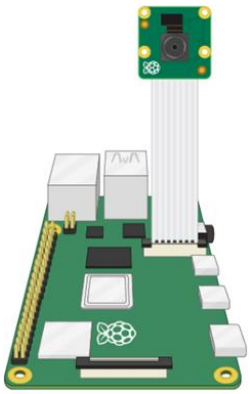
4.1. Turn of the Raspberry Pi!!!

4.2. Gently pull up on the edges of the **CSI port's** plastic slot:





4.3. Insert the Camera Module ribbon cable. Make sure the cable is the right way round:

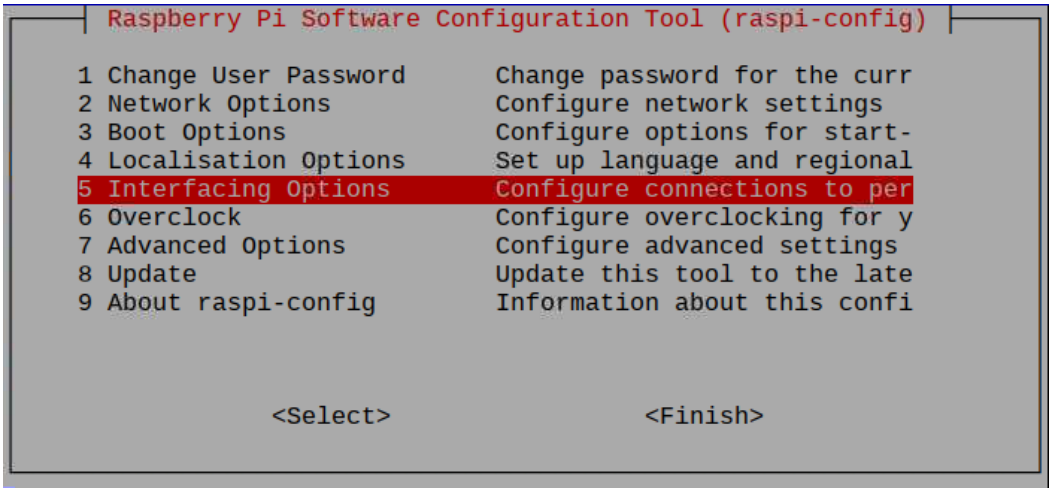


4.4. Push the plastic clip back into place:

4.5. Turn on the Pi and issue the following command to open the configuration panel:

$ sudo raspi-config

4.6. Select the *Interfacing Options:*



4.7. Enable the camera
4.8. Reboot your device

5. Testing the camera module

Before you start working, it is always good to ensure that your camera is working correctly. Otherwise, you could easily waste your time wondering when your code is not working correctly when it is simply the camera module itself that is causing you problems.

5.1. Open a Terminal and take a photo. The bellow command activates your Pi camera, displays a preview of the image, and then after a few seconds, takes a picture, and saves it to your current working directory. You can adjust the location and the size of the image (**-h** and **-w** define the height and width of the image):

$ raspistill -o / your_working_dir /*image_name.jpg* -w 640 -h 480

5.2. View the photo (if you do not have an image viewer installed on your system, issue the following command: **sudo apt-get install display**):

$ ls

$ display *image_name.jpg*

6. Install important modules

6.1. Install the *imutils* image manipulation package:

```
$ pip3 install imutils
```

6.2. Install a package that allows you to access your Pi camera from Python. While the standard **picamera** module provides methods to interface with the camera, we need the (optional) **array** sub-module in order to utilize OpenCV. OpenCV represents images as NumPy arrays — and the **array** sub-module allows us to obtain NumPy arrays from the Raspberry Pi camera module:

```
$ pip3 install "picamera[array]"
```

6.3. Install the **Numpy** numerical processing module. It may take a long time!

```
$ pip3 install numpy
```

7. Accessing Pi camera image from Python

The below code is an example about how you can capture and display images using the *picamera* module and *OpenCV (cv2)*.

7.1. Import necessary modules:

```
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2
```

7.2. Initialize the camera and create a reference to the camera capture:

```
camera = PiCamera()
capture = PiRGBArray(camera)
#allow the camera to warm up
time.sleep(1.0)
```

7.3. Take a picture in BGR (blue, green, red) format and display the image on the screen:

```
camera.capture(capture, format="bgr")
image = capture.array
#show image until a key is pressed
cv2.imshow("Image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

8. Accessing Pi camera video stream from Python

In the OpenCV library there is a *VideoCapture* function which can capture video stream from USB webcams. However, it will not work on your Pi when you would like to use your Pi camera. Fortunately, the **imutils** packet (thanks to Adrian Rosebrock) contains a unified interface to both *picamera* and *cv2.VideoCapture* using a single class named **VideoStream**. The goal of the VideoStream class is to construct a unified class to both Raspberry Pi camera and built-in/USB web cameras in PCs. Depending on the parameters supplied to the VideoStream constructor, the right video stream class (USB or picamera) will be instantiated. The following code is an example about the usage of the **VideoStream** class**.**

8.1. Import necessary modules:

```
from imutils.video import VideoStream
import imutils
import datetime
import argparse
import time
import cv2
```

8.2. Construct argument parser. Unlike other previous sample Python programs, you should run this code from the consol wher you can provide command line arguments. Now we have a single switch *–picamera* which is used to indicate whether the Pi camera or the USB camera will be used.

```
ap = argparse.ArgumentParser()
```

```
ap.add_argument("-p", "--picamera", type=int, default=-1, help="Pi
camera or the USB camera will be used")
args = vars(ap.parse_args())
```

8.3. Initialize video stream and allow the camera to warm up:

```
vs = VideoStream(usePiCamera=args["picamera"] > 0).start()
time.sleep(2.0)
```

8.4. Inside an infinite loop the program looping over frames from the video stream until we press the **q** button on the keyboard. The *read()* method returns the most recent frame from the stream. This frame will be resized the current timestamp will be drawn on it. Finally the modified frame will be displayed on the screen.

```
while True:
    # acquire a frame from the video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=400)

    # draw the timestamp on the frame
    timestamp = datetime.datetime.now()
    ts = timestamp.strftime("%A %d %B %Y %I:%M:%S%p")
    cv2.putText(frame, ts, (10, frame.shape[0] - 10),
    cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255), 1)

    # show the frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

# cleanup – close window(s) and stop video stream
cv2.destroyAllWindows()
```

```
vs.stop()
```

8.5. To try out the program, go to your working directory in the terminal and issue the below command. Note that the *--picamera 1* argument indicates that, we would like to use our Pi camera instead of a USB camera!

```
$ python3 name_of_your_file.py --picamera 1
```

9. Use ZBar for barcode recognition

If we have an image with barcode or QR code, we can pass it to a dedicated Python barcode decoding library such as the **ZBar.** The ZBar library will then detect and decode the barcode or QR code on the image. OpenCV can be used in combination with ZBar to perform any further processing and display the result.

9.1. Installing the ZBar library:

```
$ sudo apt-get install libzbar0
$ pip3 install pyzbar
```

9.2. Open a new Python file and import the necessary libraries:

```
from pyzbar import pyzbar
import cv2
```

9.3. Download a sample image. Take the sample image and pass it to the zbar's decode function:

```
image_path = "…/barcode.png"
image = cv2.imread(image_path)
barcodes = pyzbar.decode(image)
```

9.4. Extract information from the barcode variable and visualize them on the sample image:

```
for barcode in barcodes:
    # draw bounding box around the detected object
    (x, y, w, h) = barcode.rect
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)

    # the barcode data is a bytes object so if we want to draw it on
    # our output image we need to convert it to a string first
    barcodeData = barcode.data.decode("utf-8")
```

```
        barcodeType = barcode.type

        # draw the barcode description and type on the image
        text = "{} ({})".format(barcodeData, barcodeType)
        cv2.putText(image, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (0, 0, 255), 2)
```

9.5. Show the output image (**task to be solved alone!**). A sample image can be seen below.

Task to be solved alone:

9.A. Extend the above code to work with real-time video stream! Try to detect both QR codes and barcodes!

References

[1] Rosebrock, A. https://www.pyimagesearch.com/2016/04/18/install- guide-raspberry-pi-3-raspbian-jessie-opencv-3/. Accessed on 03/01/2020.

[2] Rosebrock, A. https://www.pyimagesearch.com/2018/09/19/pip-install-opencv/. Accessed on 03/01/2020.

[3] Rosebrock, A. https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/. Accessed on 03/01/2020.

[4] Rosebrock, A. https://www.pyimagesearch.com/2016/01/04/unifying-picamera-and-cv2-videocapture-into-a-single-class-with-opencv/. Accessed on 03/01/2020.

[5] Mallick, S. https://www.learnopencv.com/barcode-and-qr-code-scanner-using-zbar-and-opencv/. Accessed on 06/10/2020.

[6] Rosebrock, A. https://www.pyimagesearch.com/2018/05/21/an-opencv-barcode-and-qr-code-scanner-with-zbar/. Accessed on 06/10/2020.